領域制約の下でのゲーム木探索(計算アルゴリズムと計算量の基礎理論)

AUTHOR(S):

加藤, 芳朗; 茨木, 俊秀

CITATION:

加藤, 芳朗 ...[et al]. 領域制約の下でのゲーム木探索(計算アルゴリズムと計算量の基礎理論). 数理解析研究所講究録 1988, 666: 168-179

ISSUE DATE:

1988-07

URL:

http://hdl.handle.net/2433/100673

数理解析研究所講究録
666巻 1988年 168-179

領域制約の下でのゲーム木探索

京都大学工学部　加藤　芳朗　(Yoshiroh KATOH)
京都大学工学部　茨木　俊秀　(Toshihide IBARAKI)

あらまし

完全情報2人零和ゲームはminimaxゲーム木で表現され、ゲーム木を解くことで両者が最適な手を選んだ場合の結果を知ることができる。この目的に、$\alpha-\beta$、SSS*等の探索法が提案されている。しかし、特にSSS*を実行するためには、木の高さに関して指数オーダーの領域が必要とされる。これを緩和するために領域制御可能な探索法を提案し、さらに、オセロゲームを用いた数値実験の結果について述べる。

1. まえがき

将棋、囲碁、オセロ等の完全情報2人零和ゲームは、minimaxゲーム木によって表現され、ゲーム木を解くことにより、両者が最適な手を選んだ場合の結果を知ることができる。ゲーム木を解くには、必ずしもゲーム木のすべての節点を探索しなくてもよいことから、必要な探索領域を減少させる試みが数多くなされ、$\alpha-\beta$[3],[4]、SSS*[5]等がよく知られている。特に、Stockmanによって提案されたSSS*は、$\alpha-\beta$法にくらべて探索領域を小さくできることが知られているが、その実行において、木の高さに関して指数オーダーの記憶領域が必要とされる。このため、計算機上で実現した場合、解くことができるゲーム木の大きさが制限され、実用上問題がある。

本報告では、3章でSSS*の性質を述べた後、4章において領域制御可能な探索法を提案する。さらに、5章ではオセロゲームを用いた数値実験の結果について述べる。

2. ゲーム木とその探索

2.1 ゲーム木

図1にminimaxゲーム木の例を示す。節点はゲームの各局面に対応し、各節点の子節点は、可能な次手を打つことによって得られる局面を表わす。端末節点(terminal node)は1つの終局を表わし、その局面の静的評価値が与えられている。ゲームは、MAXおよびMINの2人の競技者によって進められ、それぞれの手番に対応する局面を、MAX節点□とMIN節点○で示す。ゲーム木では、根から下へ向かう任意の路に沿って、MAX節点とMIN節点が交互に現れる。

各節点Pに対し、その局面から両プレイヤーが最善手を選び続けたとき、実現される終局の静的評価値をPのminimax値と呼び、f(P)と記す。ゲーム木の端末節点からはじめ、その他

の節点を下から上へ次の計算にしたがって処理すれば、すべての節点Pのｆ(Ｐ)を計算することができる(図１参照)。ただし、端末節点のｆ値はその静的評価値に等しい。

$$f(P):=\begin{cases}\max\{f(Q)\mid Q\in S(P)\}, & P:\mathrm{MAX}\\ \min\{f(Q)\mid Q\in S(P)\}, & P:\mathrm{MIN}.\end{cases} \tag{2.1}$$

ここで、Ｓ(Ｐ)は節点Ｐの子節点の集合を表わす。ゲーム木の根節点をP_0とするとき、$f(P_0)$をゲーム値と呼び、$f(P_0)$を正確に求めることをゲームを解くという。図１の例では、$f(P_0)=30$である。ところで、ゲーム木が陽に与えられれば、式(2.1)によってゲーム値を計算できるが、ゲーム木は通常きわめて大規模であるため、式(2.1)の計算を直接実行することは困難である。そのため、ゲーム木の一部分だけを実際に生成することでゲーム値を計算することを目的とした、種々の探索法が提案されている。

２．２　カット条件

ゲーム木の探索中において、ある節点Ｐの子孫の探索を進めてもゲーム値$f(P_0)$に影響を与えないことが明らかになれば、Ｐを根とする部分木の探索を考慮から除外することができる。このような操作をカット(cut)という。カットを効果的に利用することにより、探索時間の短縮が可能となる。各探索法に応じてカットを実際に適用する手順は異なってくるが、カットを可能にする条件は統一的に表現することができる。

与えられた探索法において、ある時点までに生成されたゲーム木の一部を探索木といい、Ｔと記す。Ｔはゲーム木の根P_0を含み、$P(\neq P_0)$がＴに含まれるならばその親も必ず含まれるという性質をもつ部分木である。ゲーム木自身も探索木になり得るが、一般の場合と区別する

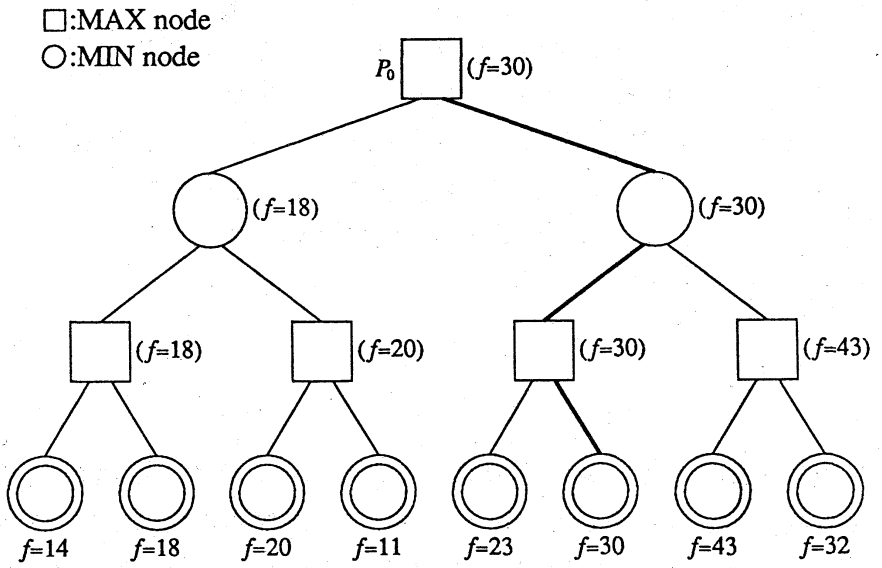

図１　ゲーム木の例(◎は端末節点を示す)

ために、ゲーム木の全体をT_0と記すことがある。探索木Tの節点Pに対し、その子節点がTに含まれていないならばPは突節点(tip node)であるという。また、突節点でなければ中間節点という。T_0の端末節点は、Tに含まれているならば必ず突節点であるが、その逆は一般には成立しない。探索木Tの節点Pに対し、f(P)の上下界値$U_b(P)$および$L_b(P)$を、Tの下方から上方へ次の計算を適用することで求めることができる。

$$U_b(P)=\begin{cases} f(P),\ P:\text{端末節点} \\ +\infty,\ P:T\text{の突節点であるが}T_0\text{の端末節点でない} \\ \max\{U_b(Q)\mid Q\in S(P)\},\ P:\text{MAX中間節点} \\ \min\{U_b(Q)\mid Q\in S(P)\},\ P:\text{MIN中間節点}. \end{cases} \tag{2.2}$$

$$L_b(P)=\begin{cases} f(P),\ P:\text{端末節点} \\ -\infty,\ P:T\text{の突節点であるが}T_0\text{の端末節点でない} \\ \max\{L_b(Q)\mid Q\in S(P)\},\ P:\text{MAX中間節点} \\ \min\{L_b(Q)\mid Q\in S(P)\},\ P:\text{MIN中間節点}. \end{cases} \tag{2.3}$$

その名が示すとおり、

$$L_b(P)\leqq f(P)\leqq U_b(P) \tag{2.4}$$

の関係が一般的に成立する。次に、Pの先祖部分の情報を集約した$U_t(P)$、$L_t(P)$を以下のように定義する。

$$U_t(P)=\min\{U_b(Q)\mid Q\in AMIN(P)\} \tag{2.5}$$

$$L_t(P)=\max\{L_b(Q)\mid Q\in AMAX(P)\} \tag{2.6}$$

ただし、AMIN(P)(AMAX(P))は節点PのMIN先祖(MAX先祖)の集合を表わす。なお、本稿では、Pの先祖あるいは子孫というとき、P自身は含まない。

以上の定義を用いて、カットの必要十分条件は、次のように書かれる[(1)]。

$$\max[L_t(P),\ L_b(P)]\geqq\min[U_t(P),\ U_b(P)] \tag{2.7}$$

この条件より、探索木Tに属しているすべての端末節点Pは、そのf値を求めると、$L_b(P)=U_b(P)=f(P)$によって、常にカット条件を満たすことがわかる。また、探索木Tにおいて、まだ調べられていない突節点Pに対するカット条件(2.7)は、

$$L_t(P)\geqq U_t(P) \tag{2.8}$$

に等しい。

2.3 探索法

ゲーム木探索の一般的な手順は、次のように書かれる。まず、根節点P_0のみからなる探索木から計算を開始する。計算の各反復時点において、探索木の突節点の中でまだ調べられていない節点(このような節点をOPEN節点と呼ぶ)をリストOPENに保持しておき、その中の1つの節点Pを選択する。そのPが予備カット条件$L_t(P)\geqq U_t(P)$を満たさなければ、Pを探索す

る。すなわち、Pが端末節点ならば、静的評価値を計算し、そうでなければPを展開し(つまり、Pの子節点をすべて生成し)、探索木を成長させる。何回かの反復後、OPENリストが空になるまでに、$U_b(P_0)=L_b(P_0)$が成立し、計算が終了する。もちろんこのとき、ゲーム値$f(P_0)$は$L_b(P_0)$に等しい。

procedure　GSEARCH(1)

G1(初期化): OPEN:={P_0}, $U_b(P_0):=\infty$, $L_b(P_0):=-\infty$。

G2(探索): $L_b(P_0)=U_b(P_0)$なら停止する。さもなければ、OPENから節点Pを1個選び、OPENから削除する。

G3(予備テスト): $L_t(P)\geqq U_t(P)$なら、G2に戻る。

G4(上下界値の計算): Pが端末節点ならば、$L_b(P):=f(P)$, $U_b(P):=f(P)$を計算し、現在の探索木のすべての節点のL_b, L_t, U_b, U_tの値を更新する。Pは必ずカット条件を満たすので、そのままG2に戻る。

G5(展開): Pの子節点Qをすべて生成し、仮に、$L_b(Q):=-\infty$, $U_b(Q):=\infty$と置いた後、すべてのQをOPENに加える。G2に戻る。 □

これまでに提案されたいろいろな探索法は、各反復において、OPENから取り出す節点の選択法の相違として捉えることができる。例えば、代表的な探索法であるα－β法は、OPENの中で、最も左に位置する節点を選ぶ探索法であり、いわゆる深さ優先探索(depth-first search)に基づくものである。α－β法の名の由来は、この場合、

$$\alpha(P)=\max_{Q\in AMAX\text{-}LS(P)} L_b(Q)$$

$$\beta(P)=\min_{Q\in AMIN\text{-}LS(P)} U_b(Q)$$

を用いると、予備カット条件(2.8)が$\alpha(P)\geqq\beta(P)$と書かれることによる。ただし、根節点P_0から節点Pに到る路を$\pi(P)$で示すとき、AMAX-LS(P)(AMIN-LS(P))は節点Pの先祖に位置するMAX節点(MIN節点)の子節点で、$\pi(P)$より左に位置する節点の集合である。

一方、ＳＳＳ*は、いわゆる最良優先探索(best-first search)に基づく探索法であり、以下のように記述される。

ＳＳＳ*: OPENの中で$\beta(P)$を最大にする節点を選ぶ(同点の場合は左の節点を優先する)。

なお、ＳＳＳ*は元の論文(5)では全く異なる形式で書かれていたが、文献(1)で上のように定式化できることが示された。このように定義されたＳＳＳ*は、その実行において、通常、木の高さに関して指数オーダーの記憶領域を必要とすることが知られている。次章でＳＳＳ*の性質を述べる。

３．ＳＳＳ*の性質

ＳＳＳ*の性質を示すために、いくつかの定義を行う。まず、ある節点Pに対して、GSEARCHのステップG4あるいはG5が適用されたとき(すなわち、Pが端末節点ならばf(P)が計算され、Pが中間節点ならばPの子節点が生成されたとき)、節点Pは探索されたといい、G5

でＰの子節点が生成されたとき、Ｐは展開されたという。このとき、任意のゲーム木に対して、

（ａ）探索法Ａは探索法Ｂが節点Ｐを探索するときに限ってＰを探索する、

（ｂ）探索法Ａは探索法Ｂが節点Ｐを展開するときに限ってＰを展開する、

が成立するとき、探索法Ａは探索法Ｂを凌ぐ(surpass)という。さらに、探索法Ａが探索法Ｂを凌ぎ、かつＡが探索(展開)しないある節点をＢが探索(展開)するようなゲーム木が少なくとも１つ存在するとき、ＡはＢを真に凌ぐという。探索法Ａを真に凌ぐ探索法が存在しないとき、Ａは非劣(unsurpassed)であると定義する。

次に、ある探索木の節点Ｐに対し、任意のＱ∈AMIN-LS(Ｐ)がカット条件を満たすとき、Ｐは左MAX認証であるという。さらに、記法、

$$\alpha(P)=\max_{Q\in AMAX\text{-}LS(P)} U_b(Q)$$

$$\beta(P)=\min_{Q\in AMIN\text{-}LS(P)} L_b(Q)$$

を導入する。節点Ｐが、

$$\beta(P)>\alpha(P) \tag{3.1}$$

を満たすならば、節点Ｐは有資格(eligible)であるという。GSEARCHのＧ２で、OPENから常に有資格な節点だけを選ぶ探索法を有資格探索といい、以下の性質が知られている。

［補題３．１］(1)　任意の有資格探索はα－β法を凌ぐ。逆に、有資格探索ではない探索法はどれもα－β法を凌がない。　□

さらに、ＳＳＳ*に関して、次の補題が知られている。

［補題３．２］(1)　ＴをＳＳＳ*によって生成された探索木とし、節点ＰをＳＳＳ*によって選ばれたOPEN節点とする。このとき、$U_t(P)>L_t(P)$が成り立てば(つまり、予備カット条件が成立しなければ)、Ｐは有資格、左MAX認証であり、かつ$\beta(P)=\bar{\beta}(P)$が成立する。　□

この補題は、ＳＳＳ*が有資格探索の特別な場合であることを意味している。したがって、補題3.1より、ＳＳＳ*はα－β を凌ぐことがわかる。

また、次の定理も知られている。

［定理３．１］(6)　ＳＳＳ*は非劣である。　□

４．領域制御可能な探索法

４．１　再帰型探索法 RSEARCH

領域制御可能な探索法として、次のような探索法のクラスを導入する。

まず、節点Ｐとある定数cを引数とし、trueあるいはfalseを返す論理型関数cond(Ｐ，c)を適当に定めておく。但し、ゲーム木の根節点P_0に対し、常に、cond(P_0，c)＝falseとする。

その後、GSEARCHと同様に探索は開始されるが、OPENリストから選ばれた節点Ｐに対して、cond(Ｐ，c)＝falseであれば、探索はそのまま進行する。しかし、cond(Ｐ，c)＝trueが成立するとき、以後の探索は節点Ｐが終端される(すなわち、ｆ(Ｐ)が計算されるか、Ｐがカット

される)まで、節点P及びPの子孫に制限される(このとき、Pの子孫に対して、異なるcond'を用いてもよい)。これは、とりあえず、Pを根とする部分ゲーム木を解くことを意味する。さらに、この制限は、探索が進行するにつれて再帰的に課せられる。節点Pが終端された後、探索はPから他の節点へ戻る。

この探索法が、あるゲーム木を解くのにどの程度の記憶量Mを必要とするかを調べよう。この探索法は、次のような探索木Tを生成する(図2参照)。つまり、再帰呼出しの深さがkであるとき、探索木Tは、P_{i-1}を根とし、$cond_i(P_i, c_i)$=trueであるような節点P_iを突節点に持つような部分探索木$T_i(1\leq i\leq k-1)$と、P_{k-1}を根とする部分探索木T_kとから成る。但し、$cond_i(1\leq i\leq k)$は、部分探索木T_iの節点(但し、T_iの根節点P_{i-1}を除く)に対して定義された関数であり、$c_i(1\leq i\leq k)$は、そのときのcの値である。当然、各$T_i(1\leq i\leq k)$は、$P_i(1\leq i\leq k-1)$及びT_kにおいて選ばれる節点P_kが変わるにつれて変化し、kの値も探索中に変動する。各T_iに対して必要とされる記憶量をspace(T_i)で表わす時、このゲーム木を解くのに必要とされる全記憶量Mは、

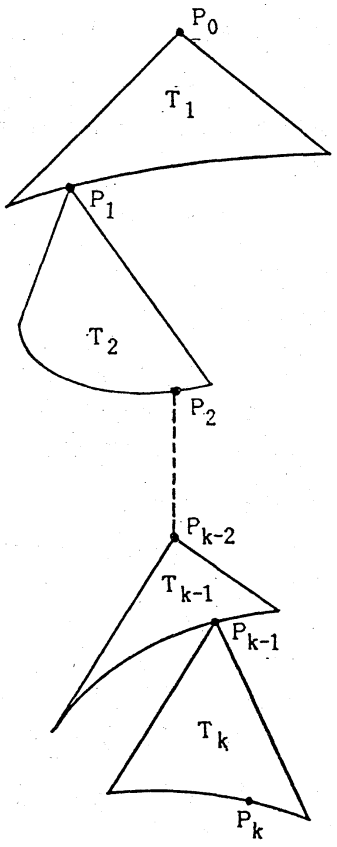

図2 RSEARCHによって生成される探索木

$$M = \max \sum_{i=1}^{k} space(T_i) \tag{4.1}$$

となる。ここで、maxは、このゲーム木の探索中に出現するすべての$T_i(1\leq i\leq k)$とkの値に対してとるものとする。

$cond_i(P, c_i)(i\geq 1)$としては、次のようなものが考えられる。

$$cond_i(P, c_i) := \begin{cases} \text{false}, & depth_i(P) < c_i \\ \text{true}, & depth_i(P) = c_i. \end{cases} \tag{4.2}$$

あるいは、

$$cond_i(P, c_i) := \begin{cases} \text{false}, & space_i(P) < c_i \\ \text{true}, & space_i(P) \geq c_i. \end{cases} \tag{4.3}$$

ここで、$depth_i(P)$は現在の部分探索木におけるPの深さを、$space_i(P)$はPが属する現在の部分探索木が必要とする記憶量を表わす。

この探索法の形式的な記述は以下のようになる。ただし、$\Phi(P)=\{L_b(P), U_b(P), L_t(P), U_t(P), \alpha(P), \beta(P), \bar{\alpha}(P), \bar{\beta}(P)\}$であり、計算に必要なデータを示す。特に、$\Phi(P_0)=\{L_b(P_0), U_b(P_0), -\infty, \infty, -\infty, \infty, -\infty, \infty\}$である。

procedure　RSEARCH

$L_b(P_0):=-\infty$、$U_b(P_0):=\infty$と置いた後、solve(P_0, $cond_1$, c_1, $\Phi(P_0)$)を呼ぶ。ここで、P_0はゲーム木の根節点、$cond_1$は適当に決められた関数、c_1は適当に決められた定数である。solveからreturnしたとき、ゲーム値$f(P_0)=L_b(P_0)$が成立するので、停止する。

procedure　solve(P', cond, c, Φ(P'))

R１(初期化): OPEN:={P'}。

R２(探索): $\max[L_t(P'), L_b(P')] \geqq \min[U_t(P'), U_b(P')]$ならば、return。さもなければ、OPENから節点Pを１個選び、PをOPENから削除する。

R３(予備テスト): $L_t(P) \geqq U_t(P)$なら、Ｒ２に戻る。

R４(上下界値の計算):

１) Pが端末節点ならば、$f(P)$を計算し、$L_b(P):=f(P)$, $U_b(P):=f(P)$とする。

２) cond(P, c)＝true ならば、関数cond'と定数c'を適当に決めて、solve(P, cond', c', Φ(P))を呼ぶ。

１)あるいは２)の場合、現在のP'を根とする部分探索木のすべての節点のL_b, U_b等の値を更新する。また、このとき、Pは必ずカット条件を満たすので、そのままＲ２に戻る。

R５(展開): Pの子節点Qをすべて生成し、仮に、$L_b(Q):=-\infty$, $U_b(Q):=\infty$と置いた後、すべてのQをOPENに加える。Ｒ２に戻る。　□

RSEARCHは、ゲーム木が有限ならば、有限回の反復で停止し、正しくゲーム値を計算する。このとき、利用可能な記憶領域が制限されているという制約の下で、できるだけ効率のよい探索法を考えたい。すなわち、現在利用できる記憶領域に照らして、関数$cond_i$ ($i \geqq 1$)、及びc_i ($i \geqq 1$)の値を適切に決定してやることによって、利用可能な記憶領域の有効利用をはかりたい。

４．２　再帰型ＳＳＳ*

RSEARCHに属する探索法の１つとして、再帰型ＳＳＳ*を次のように定義する。

再帰型ＳＳＳ*：Ｒ２において、OPENの中から$\beta(P)$を最大にする節点を選ぶ(同点の場合は左の節点を優先する)。

特に、$cond_i$＝cond ($i \geqq 1$)であり、かつcondを(4.2)で定める場合、再帰型ＳＳＳ*は、c_1をゲーム木の高さ以上に大きくとれば、通常のＳＳＳ*になり、$c_i=1$ ($i \geqq 1$)とすれば、$\alpha-\beta$になる。

再帰型ＳＳＳ*は、以下の性質を持つ。

[補題４．１][1][2]　Ｔを再帰型ＳＳＳ*によって生成された探索木とし、節点Pを再帰型ＳＳＳ*によって選ばれたOPEN節点とする。このとき、$U_t(P) > L_t(P)$が成り立てば(つま

り、予備カット条件が成立しなければ)、Pは左MAX認証であり、かつ$\beta(P)=\beta(P)$が成立する。しかし、Pが有資格であるとは限らない。　□

この補題から、残念ながら、再帰型ＳＳＳ*は一般には有資格探索ではないことが結論される。

４．３　再帰型ＳＳＳ*の記憶領域制御

まず、特に、$cond_i=cond(i\geq 1)$であり、かつcondを(4.2)で定める場合の再帰型ＳＳＳ*が、与えられた記憶量内で正常に動作するような$c_i=c(i\geq 1)$の決定法について考える。この場合、任意のゲーム木に対して、その高さをnとすれば、(4.1)の全記憶量Mは、

$$M \leq \lceil n/c \rceil \times space(T^c)$$

を満たす。ここで、$space(T^c)$は、ＳＳＳ*が高さcの部分ゲーム木を解くのに必要とする最悪記憶量を表わす。$space(T^c)$の実際の評価には、個々のゲームに対する計算結果を利用することになるが、ゲーム固有のbranching factor bを用いて、

$$space(T^c) \approx b^c$$

となることが予想される。$space(T^c)$の評価が出来たとすると、実際の計算において利用できる記憶量をM'とするとき、

$$\lceil n/c \rceil \times space(T^c) \leq M'$$

となるように適当なcを選べばよい。この場合、制約条件の範囲内で、cの値をできるだけ大きくとれば、計算時間の最小化がはかれることが数値実験により確かめられている(５章参照)。しかし、常に、最悪の場合に必要とされる記憶領域を確保しておくのは効率的ではない。そこで、$cond_i$、及びc_iの値を探索中に、適応的に変化させる方法が考えられ、やはり、５章で、数値実験に基づいて考察を加える。

次に、特に、$cond_i=cond(i\geq 1)$であり、かつcondを(4.3)で定める場合の再帰型ＳＳＳ*について考える。このときは、任意のゲーム木(その高さをnとする)に対して、最悪の場合を考えて、実際の計算において利用できる記憶量をM'とするとき、

$$\sum_{i=1}^{k}(c_i+d-1) \leq M'$$

となるように適当な$c_i(1\leq i\leq k)$を選べばよい。ただし、kは再帰呼出しの最大の深さ(高々n-1である)、dはゲーム木の節点の子の数の最大値である。

５．オセロゲームによる実験結果

本章では、再帰型ＳＳＳ*をオセロゲームに適用した結果について述べる。なお、使用された計算機はSun microsystems社のワークステーションSUN-3 52M(1.5MIPSの性能を有する)であ

り、プログラム記述言語はCである。

実験は、6×6オセロゲームの中盤局面(残り手数 16手)をランダムに10個発生し、再帰型SSS*でその局面を解き、それらの平均を求めた。すなわち、実験に用いたゲーム木の高さはすべて16である。

第1の実験は、$cond_i = cond(i \geq 1)$、かつcondを(4.2)のように定め、$c_i = c(i \geq 1)$とした。結果を図3に示す。c=2~16までの結果は、文献(7)のプログラムを、本目的に修正したプログラムによる結果であり、c=1(α−β法)の結果は、α−β法(ただし、節点の順序付けは優先順位表による)専用のプログラムによる結果である。図3において、節点数とは、探索された端末節点数を意味し、MAX CELLは、探索中に探索木が最も大きくなった時に必要としたCELLの個数(すなわち、最大の探索木の全節点数)を表わす。図3から、再帰型SSS*は、SSS*に近づくほど、必要とされる記憶領域は激増するが、効率はよくなっている。

なお、実験1のc_iの定め方には、種々の変型も考えられる。その1つとして、$c_1 = c$、$c_2 = 16 - c$の場合について実験を行なったが(c<8の場合、実験1と異なる結果を与える)、実験1に比べてかなり悪い結果となった。

次に、与えられた記憶量内で、できるだけ効率のよい探索法を実現するために、計算中に関数$cond_i$、及びc_iの値を動的に変化させることを試み、実験2と実験3を行なった。

実験2では、$cond_i(i \geq 1)$を次のように定めた。

$$cond_1(P, c_1) := \begin{cases} \text{false}, & N < c_1 \\ \text{true}, & N \geq c_1. \end{cases}$$

$$cond_i(P, c_i) := \begin{cases} \text{false}, & depth_i(P) < c_i \\ \text{true}, & depth_i(P) = c_i. \end{cases} \qquad (i \geq 2)$$

ここで、Nは現在、実際に使われているCELLの個数を表わす。実際に実験を行なったのは、$c_i = 2 (i \geq 2)$と固定し、$c_1 = 40000, 10000, 2500$の場合である。すなわち、与えられた記憶領域をほとんど使い切るまでSSS*を実行し、記憶領域が残り僅かになったとき、α−β法的な探索に移行する場合の効率を調べた。結果は、表1に示されている。図3の結果と合わせると、これらの探索法は、いずれも、満足できるものではない。非効率さの理由は、SSS*が領域c_1を使い切ったときの探索木が、ある突節点は大きな深さを持つが小さな深さを持つものも存在するという、いびつな形状であるため、結果として、小さな深さを持つ節点にα−β法が適用されることになるためであると考えられる。その点を改善するため、次の実験を行なった。

実験3では、$cond_1$を以下のように定め、$cond_i(i \geq 2)$については、実験2と同じにした。さらに、この実験では、特に、c_1を定数ベクトル、すなわち$c_1 = (c_{11}, c_{12})$と拡張した。

$$cond_1(P, c_1) := \begin{cases} \text{false}, & N < c_{11} \text{ and } depth_1(P) < c_{12} \\ \text{true}, & N \geq c_{11} \text{ or } depth_1(P) = c_{12}. \end{cases}$$

ここで、$depth_1(P)$はゲーム木の根節点P_0からの深さを表わす。すなわち、探索の切り換え

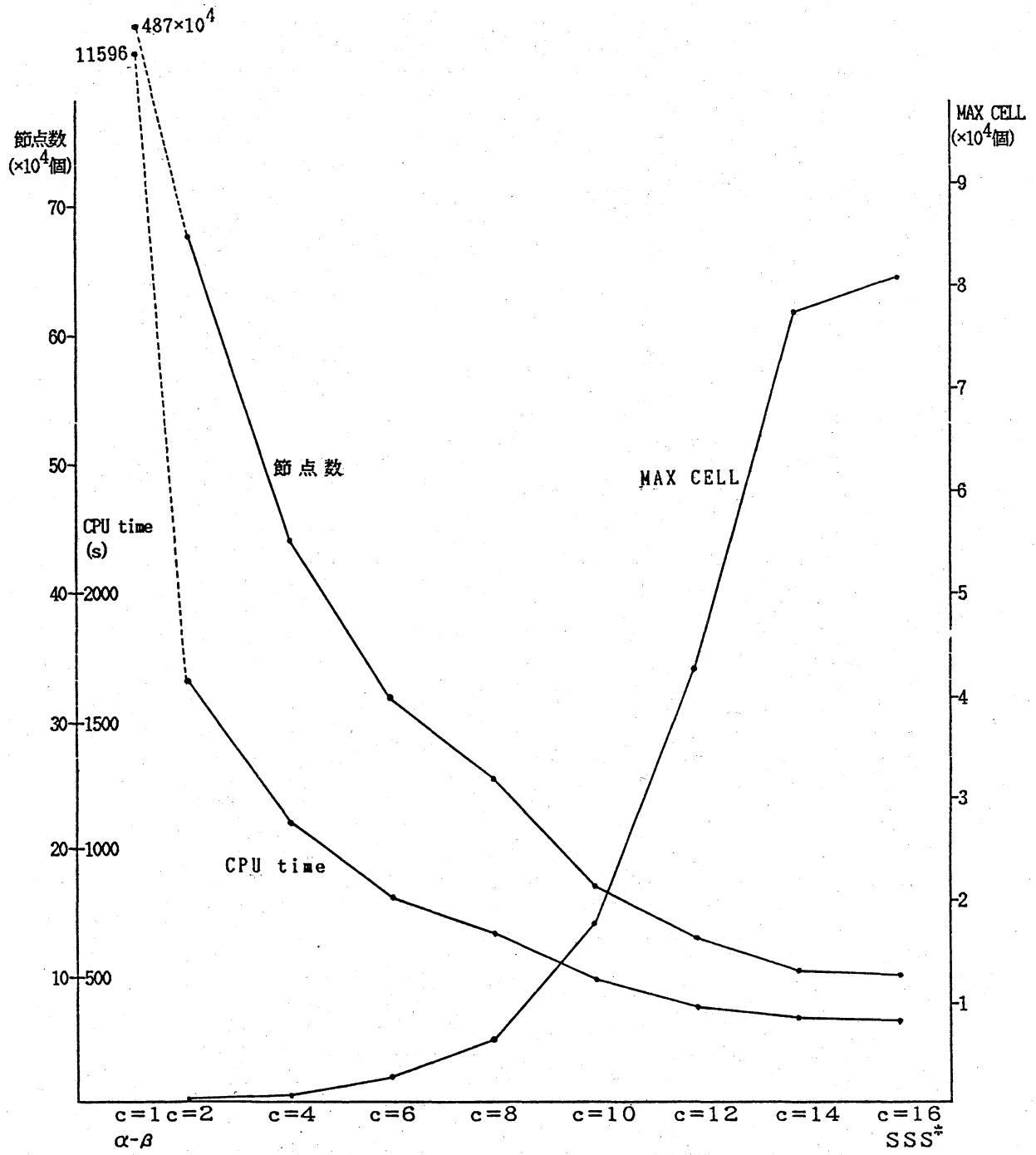

図３　第１の実験の結果（$c = c_i\ (i \geq 0)$の場合）

	time (s)	節点数 (個)	MAX CELL (個)
$c_1 = 40,000$	1,017	411,395	39,648
$c_1 = 10,000$	1,459	609,717	10,204
$c_1 = 2,500$	1,579	663,157	2,700

表 1　　実験 2 の結果 ($c_i = 2 (i \geq 2)$ の場合)

$c_{11} = 40,000$			
	times (s)	節点数 (個)	MAX CELL (個)
$c_{12} = 12$	634	248,240	35,162
$c_{12} = 10$	480	187,371	17,730
$c_{12} = 8$	728	311,055	5,952
$c_{11} = 10,000$			
	times (s)	節点数 (個)	MAX CELL (個)
$c_{12} = 10$	1,017	419,884	10,124
$c_{12} = 8$	728	311,055	5,952
$c_{12} = 6$	1,030	441,375	1,651
$c_{11} = 2,500$			
	times (s)	節点数 (個)	MAX CELL (個)
$c_{12} = 8$	1,298	550,745	2,680
$c_{12} = 6$	1,025	441,375	1,651
$c_{12} = 4$	1,356	573,199	464

表 2　　実験 3 の結果 ($c_i = 2 (i \geq 2)$ の場合)

を、使用領域Nだけではなく、探索節点Pの深さも利用して行なっている。この実験結果を、表2に示す。表1に比べて、かなり良好の結果を与えていることがわかる。

6. むすび

本報告では、ゲーム木としていわゆる静的モデル(static model)を仮定した。しかし、これらの議論は、情報付きモデル(informed model)[1]に容易に拡張できる。

ところで、従来の研究では、ゲーム木探索法の効率の比較は、端末節点の静的評価値が一様乱数によって与えられると仮定したモデル等、実際的でないモデルを確率論的に解析することによってなされたものが多く、実際のゲームに適用した結果は、あまり知られていない。そこで、本報告では、オセロゲームを使って数値実験を行ない、探索法を評価した。さらに、現実の計算においては、必要な記憶領域に関する制約を考慮しなければならないので、この目的に適した探索法の枠組としてRSEARCHを提案した。特に、再帰型ＳＳＳ*に対しては、探索の実現法に対するある程度の指針を得ることができたと思われる。しかし、RSEARCHに属する探索法に対し、関数$cond_i$(i≧1)とc_i(i≧1)の値を変えることによって、本報告で検討したもの以外にも、数多くの変型が考えられる。それらの効率の比較は今後の課題である。

謝辞

日頃ご討論いただく研究室の諸氏に感謝致します。

なお、本研究は一部、文部省科学研究費によるものである。

文献

(1) T. Ibaraki: "Generalization of alpha-beta and SSS* search procedures", Art. Int., 29, 73-117(1986).

(2) Y. Katoh: "Analysis of game solving procedure SSS* and its variants", Master Thesis, Fac. of Eng., Kyoto Univ.(1988).

(3) D. Knuth and R. Moore: "An analysis of alpha-beta pruning", Art. Int., 6, 293-326(1975).

(4) J. R. Slagle and J. K. Dixon: "Experiments with some programs that search game trees", J. ACM, 16, 189-207(1969).

(5) G. Stockman: "A minimax algorithm better than alpha-beta ?", Art. Int., 12, 179-196(1979).

(6) 加藤，茨木："ゲーム木探索法ＳＳＳ*の非劣性について"，信学論(D)，J70-D，12，PP. 2630-2639(昭62-12).

(7) 森田，国枝，津田："思考ゲームプログラミング：オセロゲームのアルゴリズムと作成法"，アスキー出版局(昭61).